SAM Managed Cache and Proposed Improvements

CD/REX/SUP
(Direct comments to Adam Lyon, lyon@fnal.gov and Robert Illingworth, illingwo@fnal.gov)
October 25, 2010

Introduction

This note describes how SAM managed cache works and includes a proposal for some minor improvements for increased efficiency and performance. This document was written as a result of thinking about how SAM managed cache could be introduced at CDF, but the improvements would benefit D0 and Intensity Frontier experiments as well.

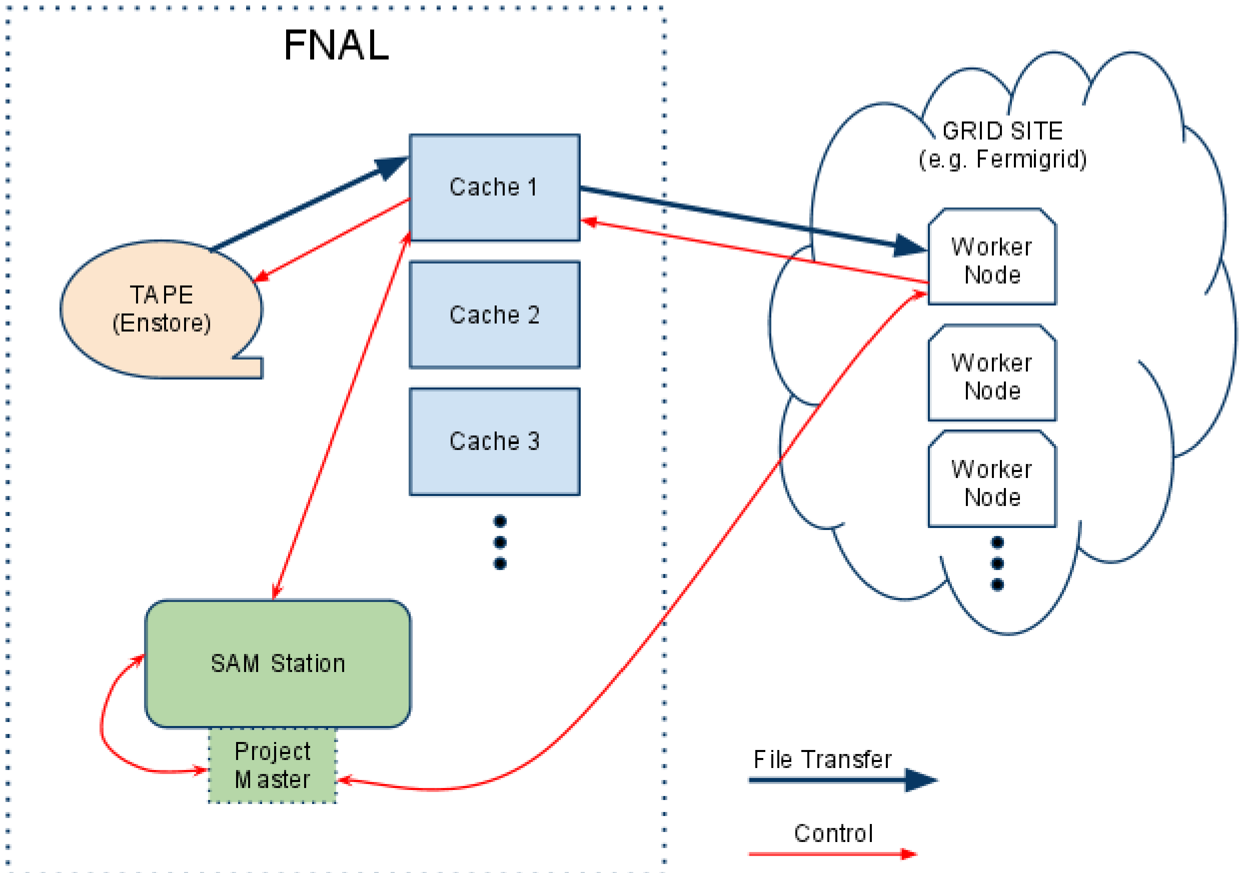
DØ has been utilizing SAM managed cache ever since RunII began. The system has reached a steady-state of very low (but non-zero) operations and maintenance load. The system is robust enough that the loss of a cache node is automatically detected and the system automatically reconfigured to avoid using it. Downtimes require no action from operators, as the system will gracefully restart automatically from a reboot. At DØ, the system delivers files at sustained rate of ~ 1 PB/week with no human intervention except to add or retire cache nodes.

Implementation

The implementation is very similar to the data handling part of DØ's SAMGrid system and is described here.

Basic system

A schematic of the system components and lines of communication are shown below.



The SAM Station is the manager of all file transfers and is an application that runs on a dedicated node (or virtual machine; DØ runs two very busy stations on one virtual machine). The Project Master is a process running on the station node and handles tracking and communication for a particular project (group of jobs processing files from the same dataset). Each project has its own Project Master running on the station node.

The Cache nodes are dedicated nodes with large disks acting as caches of data files. Each cache node runs a SAM Stager process (the SAM stager is the process that performs the file transfers).

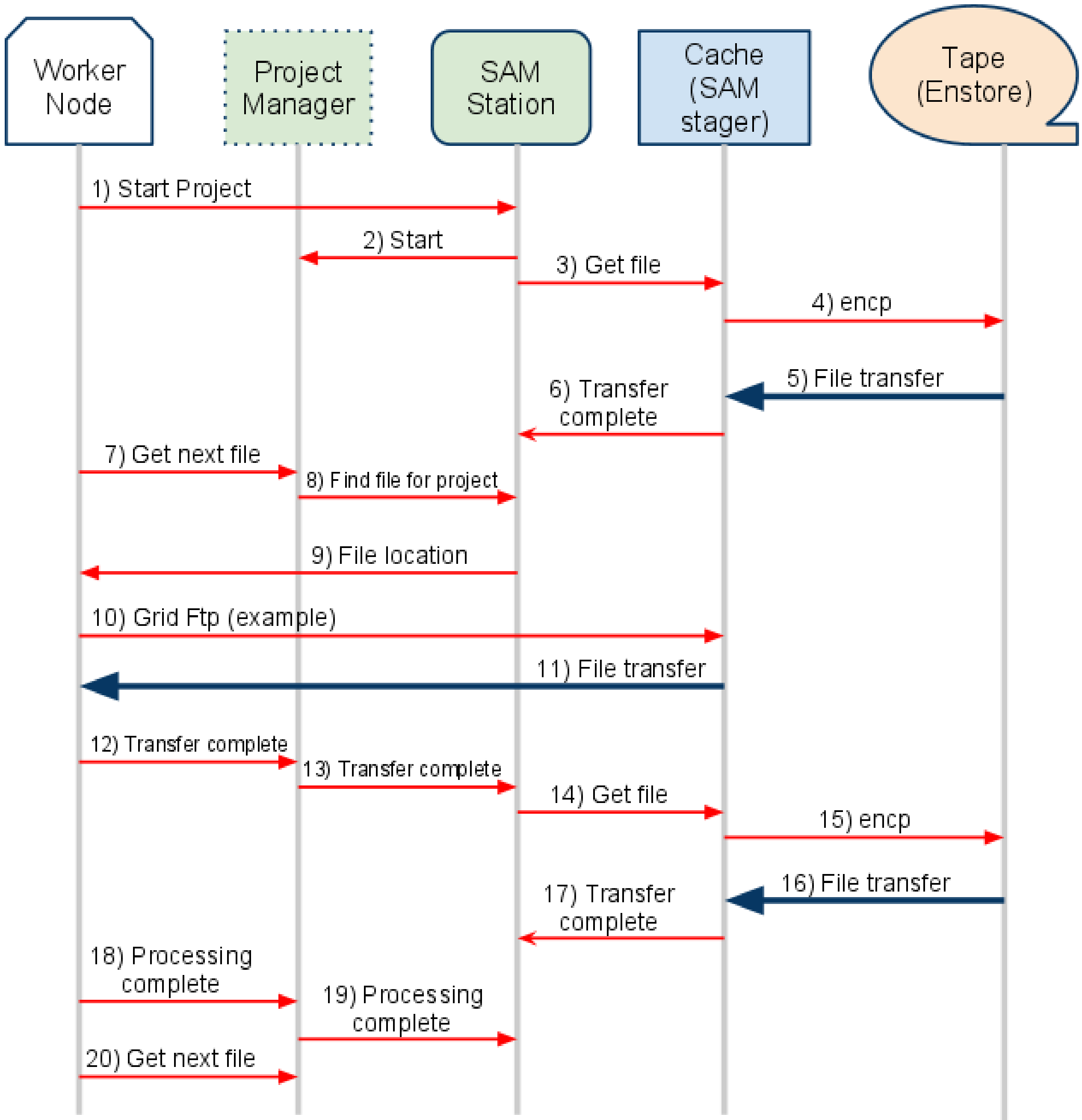
It is advisable to have the SAM station resident at Fermilab for ease of management, though that is not an important requirement. DØ has its station for OSG jobs at Fermilab. The station servicing LCG jobs is in Lyon, France.

There must be a main set of cache nodes at Fermilab for access to enstore (we do not allow direct access to enstore from offsite). It is also possible to have an additional layer of cache nodes residing closer to the remote sites (including storage elements managed with SRM). If the grid site is Fermigrid, then an additional cache layer is not needed as the worker nodes would have fast local access to the main SAM cache nodes. Offsite cache nodes can dramatically improve job efficiency for remote jobs, though the operations load is also increased, especially for sites with poor local maintenance.

The worker nodes are the nodes at a grid site running the user's job. There is no SAM cache or

SAM stager on the worker node (unlike the DØ CAB analysis farm which does have SAM cache on each worker and each worker runs a SAM stager; worker nodes used on OSG/LCG for SAMGrid are more like the situation described here and run no permanent SAM applications). DØ SAMGrid can utilize sites that block unsolicited incoming connections to worker nodes because it uses the SAM python API that does polling instead of callbacks. The SAM C++ API (used by diskcache_i - the piece of the CDF framework interfaced to SAM that deals with data handling) only uses callbacks, so only sites that allow incoming connections could be used for CDF. Extending the SAM C++ API to do polling would be a development project.

Below is a timeline of communication between these various components. Here, we assume that a job lands on a worker node and starts the project (if it hasn't been started already). Alternatively, the project may be started at job submission time (though that is inadvisable if there may be a long wait time before the jobs start on the worker nodes). We will follow the steps to deliver the first file and prepare the second (time progresses from top to bottom).



These steps are explained here:
1 & 2) In this scenario, the first job of the set starts the project. The project start can be handled other ways.

3) Once the project starts, the station immediately starts populating its cache with files necessary for the project. The number of files to prefetch is set by the SAM station configuration and current load on the system. Although in the diagram it looks like the station fetching files is synchronous, it is in fact completely asynchronous. That is the station is continuously fetching files into the cache, within limits set in the configuration, regardless of the activity of the jobs.

In this scenario, the SAM station decides to prefetch a file that is not in any of the cache disks. It instructs the SAM stager on the destination cache node to fetch the file from tape.

4) The SAM stager on the cache machine issues the encp command to fetch the file from tape.

5) The file is delivered and placed in the SAM cache.

6) The stager tells the station that the file transfer completed. The station may now seek to transfer other files depending on limits and conditions (e.g. step 14 could occur earlier than is shown in the diagram).

7) By this time (or perhaps before - remember that populating the cache is an asynchronous operation), the job is running on the worker node and needs a file to process. It issues the "sam get next file" command which is accepted by the project master process.

8) The project master notifies the station that it needs a file for a worker (a SAM process).

9) The station returns the location of the file (pointing to its path on the cache) to the job on the worker node.

10) The job code (diskcache_i for CDF) determines how to retrieve the file from the cache. In this scenario, gridftp is used and the command to copy the file from the cache node and the command is issued. See below for an idea of improvement for this process.

11) The file is transferred to the worker node via gridftp.

12 & 13) When the file arrives, the worker node notifies the project master, which in turn notifies the station that the file transfer is complete. [this step is NEW - see below]

Once the SAM station knows that the transfer is complete, it will decrement the use count on that file in the cache. If the use count goes to zero, then the file is available for deletion at some time in the future.

14) In this scenario, conditions are such that the station seeks to put another file for the project into the cache (remember, this is an asynchronous process and it could happen earlier).

15, 16, 17) In this scenario, a file is prefetched from tape into the cache.

18 & 19) From step 12 above, the worker node has been processing the file it was given. When the processing is done, it notifies the project master which notifies the station. Though this step is not really part of data handling, it is part of SAM's tracking of process success. If this processing complete signal is not issued, SAM will mark the process as failed. A recovery job can then be created using this information to reprocess the files from failed jobs.

20) Once processing a file is complete, the worker needs another file and asks for one. The process begins again at step 7.

Almost all of the code for this procedure already exists and we have much experience running this way with DØ SAMGrid on OSG/LCG. A piece that is missing that we would like to add corresponds to steps 12 & 13. Currently, SAM only receives a signal with the worker node completes **processing** a file. Having an additional signal for when the transfer completes adds several important features:

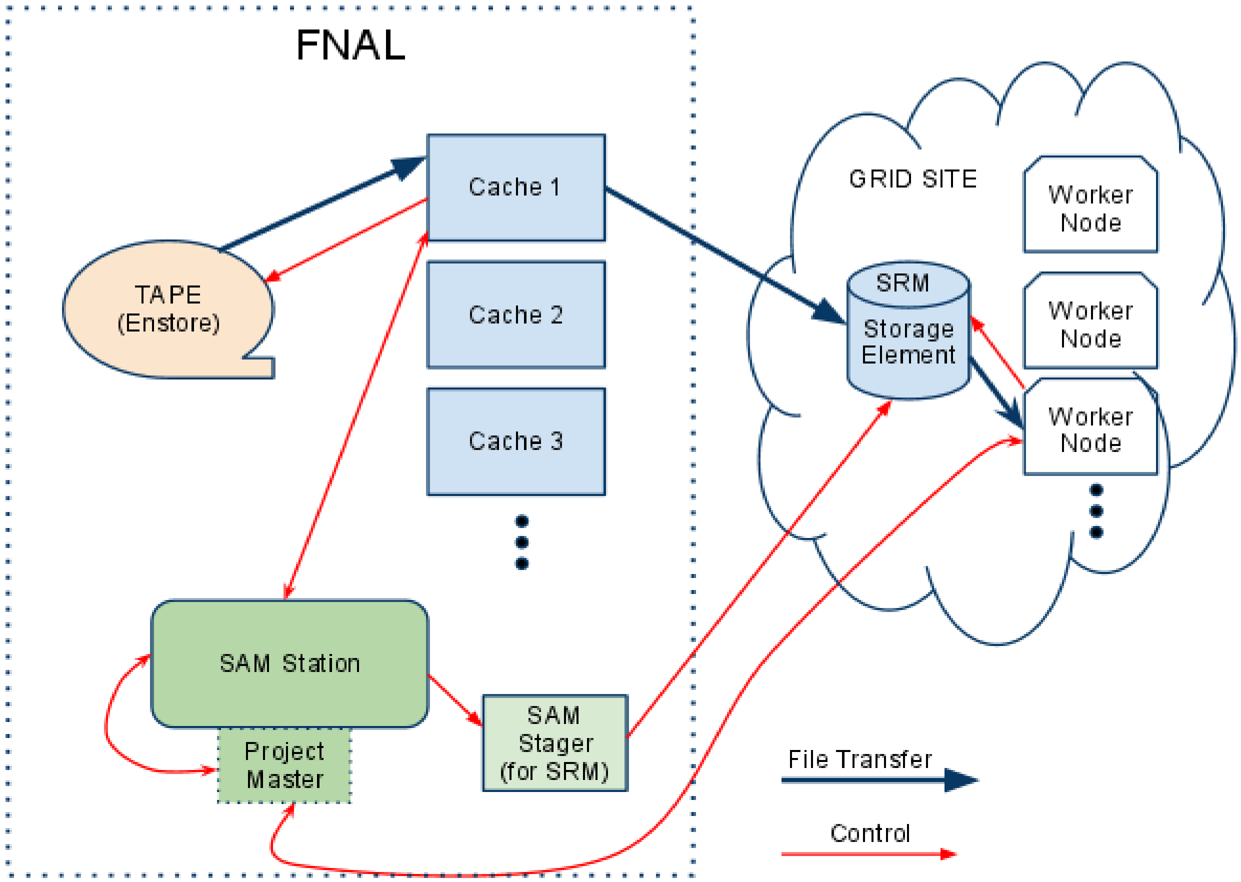
- Since the file is now on the worker node and if no one else is transferring it, it may be marked as available for deletion from the cache immediately after the transfer completes (instead of waiting for processing to finish). In practice with a large cache and long cache lifetime, this change would have little effect.

- With this signal, SAM would have a complete picture of all of the file transfers and could handle cache load balancing in a very smart way, like it does for the DØ CAB analysis farm [this signal does not exist there, but since each worker node in DØ CAB has a stager and cache, the file transfers to the workers are cache-to-cache transfers already under complete SAM control; in the case described here the job on the worker node itself is transferring the file from cache, not SAM, and so without this signal SAM does not know when the transfer ends and does not have an accurate picture of all the transfers from the cache nodes]. This information would allow SAM to count and throttle the transfers to and from the cache to keep the usability and efficiency level high. SAM could also prioritize transfers to occur from more lightly used cache disks first, thus evening out the load. Furthermore, we would also eliminate our dependence on *fcp* to throttle cache transfers.

We also envision a change in step 7 (though perhaps not at first), when the worker node asks for the next file. In the current scenario, the CDF framework piece diskcache_i somehow has to figure out how to transfer the file from the cache to its worker node. For simplicity, the transfer could always use gridftp, though for jobs running at Fermilab it could instead do a local copy with rcp/scp. Instead of putting this logic in diskcache_i, one could pass this information to SAM as a protocol hint in the “sam get next file” command so that the SAM station returns an appropriate file location that diskcache_i could just use with very little logic. For example, the job script could contain logic (perhaps hard-coded by the submission system) -- if it is running on Fermigrid, it could pass SAM the scp hint so that SAM would return a location suitable for scp. For jobs running offsite, it could give a gsi hint so that SAM will return a gridftp URL.

Utilizing Grid Storage Elements

A lesson that DØ learned is that when running jobs offsite, it is extremely important to have those jobs retrieve their files from storage local to the site. Retrieving files directly from Fermilab or some other non-local cache location reduced the efficiency of the jobs. Here, we describe how SAM utilizes grid Storage Elements (SEs). This functionality already exists and is in heavy use by DØ SAMGrid. It is, however, only as reliable as the Storage Elements and the SRM implementation. A few sites do a poor job of keeping their SE’s and SRM’s operating leading to an increased operations load on REX.



Here we see the scenario for utilizing a local storage element at the grid site via its SRM interface. The SAM station already has code for communicating with the SRM interface and moving files to the SE. An additional component needed on the SAM side is an extra SAM stager process that handles the cache-to-cache copy from a SAM cache node to the SE. The stager also issues the necessary SRM commands to the SE for file reservation, copy, and deletion. This stager could be on the station node or some other node (even outside of Fermilab). For DØ, the stager for OSG SE's runs at Fermilab while for the LCG it runs at Manchester, UK. It may be advantageous to have the stager running from its own separate station to isolate the onsite operations from the offsite.

DØ utilizes local SEs at US sites for OSG and European sites for LCG. This model works quite well.

As seen in the diagram, a file necessary for the project is fetched from tape to the Fermilab cache. The SAM stager would then initiate the srmcp call to copy that file from Fermilab to the SE. Once at the SE, and needed by the job, the station would respond to a "sam get next file" command with the location URL of the file within the SE. diskcache_i (or alternative mentioned above) would copy the file from the SE to the worker node for processing.

Development

There is some development work necessary.

- The addition of the “Transfer complete” signal from the worker node to the project master
- The addition of the “Transfer complete” signal to diskcache_i
- File protocol choice logic in diskcache_i (or just do gridftp all the time at first)
- Load balancing logic for the cache nodes (most of which is already in the code)
- Adding polling to the SAM C++ API (may not be necessary if CDF uses sites that allow incoming connections to their worker nodes).

Later, we may want to do the development for the “sam get next file” protocol hint as described above.

Robert estimates a 3-4 weeks to make these changes while performing his other duties.

Adaptation for Intensity Frontier Experiments

Intensity frontier experiments with a large data-volume (e.g. NOvA) could use this system as described above. Low data-volume experiments that could keep all of their data on disk would still benefit from a system like the above for processing on remote sites. Running locally (e.g. on Fermigrid, would not require a caching system at all, unless the demand on the data storage was too high).

In every scenario, each experiment needs a module of their framework capable of communicating with SAM, either through the C++ or Python API. For CDF and DØ, development of such a module was very complicated and time-consuming. The system described here is actually simpler than what is currently implemented at CDF and DØ, and so development should probably take only a few weeks. The REX/SAM team should have very close involvement with the experiment developer of the module. If several experiments shared the same framework, then of course the over development time would be much reduced.

For the low data-volume experiments, FermiGrid has already set up an SRM system that exposes the experiment’s Blue-Arc disk as a Grid Storage Element. If the data access load is expected to be light, then SAM could mediate transfers from the SE direct to remote worker nodes (or to SEs on remote sites). In the case of high data access load, cache disks could be deployed at Fermilab in front of the Blue-Arc and a system very similar to the one described for CDF could be deployed, with the Blue-Arc replacing the tape system in the description.

Adaptation to improve data handling at DØ

Performing the development on the “transfer complete” signal has benefits for DØ. As mentioned, we could remove our dependence on *fcp*. Though *fcp* works quite well, it sometimes has reliability issues (the servers die). Using *fcp* controls the data rate on an individual cache server basis. With the development described here, SAM could have a global view of the data access load and would be able to prioritize transfers to make the system more efficient.